1 Set of functions in QSutils package

```
Collapse <-
function(seqs){
    cls <- class(seqs)
    sqtbl <- sort(table(as.character(seqs)),decreasing=TRUE)
    seqs <- names(sqtbl)
    names(seqs) <- 1:length(seqs)
    nr <- as.integer(sqtbl)
    if(cls=="DNAStringSet") seqs <- DNAStringSet(seqs)
    if(cls=="AAStringSet") seqs <- AAStringSet(seqs)
    return(list(nr=nr,hseqs=seqs))
}
```

```
ConsSeq <-
function(seqs,w=NULL){
    if(class(seqs)!="DNAStringSet" & class(seqs)!="AAStringSet")
        stop("The input object must be a DNAStringSet or AAStringSet \n")
    if(is.null(w)) w<-rep(1,length(seqs))
    if(length(seqs)!=length(w))
        stop("The input objects must have the same length \n")
    bnms <- DNA_BASES
    if(class(seqs)=="AAStringSet")
        bnms <- AA_ALPHABET
    ntm <- FreqMat(seqs,w)
    get.nt <- function(x){
        idx <- which(x==max(x))
        if(length(idx)<2) return(idx)
        return( sample(idx,1) )
    }
    imx <- apply(ntm,2,get.nt)
    return(paste(bnms[imx],collapse=""))
}
```

```
CorrectGapsAndNs <-
function(hseqs,ref.seq){
    if(class(hseqs)!="DNAStringSet" & class(hseqs)!="AAStringSet")
        stop("The input object hseqs must be DNAStringSet or AAStringSet\n")
    if(class(ref.seq)!="character" & class(ref.seq)!="DNAString" &
    class(ref.seq)!="AAString")
        stop("The input object ref.seq must be of class character\n")
    CorrPos <- function(v,nt){
        fl <- v %in% c("-","N")
        v[fl] <- nt
        return (v)
    }
    rf <- strsplit(as.character(ref.seq),split ="" )[[1]]
```

```
    ntm <- as.matrix(hseqs)
    ntm <- sapply (1: length (rf), function (j) CorrPos(ntm[,j],rf[j]))
    if(class(hseqs)=="DNAStringSet")
        correctseq <- Biostrings::DNAStringSet(apply(ntm,1,paste,collapse=""))
    if(class(hseqs)=="AAStringSet")
        correctseq <- Biostrings::AAStringSet(apply(ntm,1,paste,collapse=""))
    return (correctseq)
}
```

```
DBrule <-
function(grpDist,hr,oDist,g.names=NULL){
    geovar <- function (D){
    if(nrow(D)<2)
        return(0)
    return(sum(D^2)/(2*nrow(D)^2))
    }
    phi <- function (d){
        sum(d^2)/ length(d)}
    g <- max(as.integer(hr))
    grpDist <- as.matrix(grpDist)
    PHI2 <- vector(mode="numeric",length=g)
    names(PHI2) <- paste("Phi2",1:g,sep=".")
    if(!is.null(g.names))
    names(PHI2) <- paste("Phi2",g.names,sep=".")
    for(i in 1:g){
        D <- grpDist[hr==i,hr==i,drop=FALSE]
        V <- geovar(D)
        PHI2[i] <- phi(oDist[hr==i])-V
    }
    idx <- match(min(PHI2),PHI2)
    clustName <- ifelse(!is.null(g.names),g.names[idx],idx)
    output <- list(Phi2=PHI2,DB.rule=idx,Type=clustName)
    return(output)
}
```

```
Diverge <-
function(vm,seq){
    if(class(seq)!="character")
        seq <- as.character(seq)
    if( !all(strsplit(seq,"")[[1]] %in% DNA_BASES))
        stop("The seq argument must be a DNA sequence")
    if(class(vm)!="numeric" & class(vm)!="integer")
        stop("The vm argument must be numeric")
    mutate <- function(nt){
        nt.nms <- DNA_BASES
        pnt <- rep(1/3,4)
        names(pnt) <- nt.nms
```

```
            fnt <- pnt; fnt[nt] <- 0
            return(sample(nt.nms,size=1,prob=fnt))
        }
        ntv <- strsplit(seq,split="")[[1]]
        len <- length(ntv)
        nm <- length(vm)
        ipos <- sample(len,size=max(vm),replace=FALSE)
        nt.var <- sapply(ntv[ipos],mutate)
        dseq <- character(nm)
        for(i in 1:nm){
            mseq <- ntv
            mseq[ipos[1:vm[i]]] <- nt.var[1:vm[i]]
            dseq[i] <- paste(mseq,collapse="")
        }
        return(dseq)
    }
```

```
    DNA.dist <-
    function(seqs,model="raw",gamma=FALSE,pairwise.deletion=FALSE){
        if(class(seqs)!="DNAStringSet")
            stop("The input object must be DNAStringSet \n")
        strm <- as.DNAbin(ape::as.alignment(as.matrix(seqs)),pairwise.deletion)
        dst <- dist.dna(strm,model=model,gamma=gamma)
        dst[is.na(dst)] <- 0
        return(dst)
    }
```

```
    DottedAlignment <-
    function(hseqs){
        if(class(hseqs)=="character")
            hseqs <- DNAStringSet(hseqs)
        if(class(hseqs)!="DNAStringSet" & class(hseqs)!="AAStringSet")
            stop("The input object must be DNAStringSet or AAStringSet \n")
        bpm <- as.matrix(hseqs)
        master <- bpm[1,]
        bpm.dot <- t(apply(bpm[-1,],1,function(x) { x[x==master] <- "."; x }))
        bpm.dot <- rbind(master,bpm.dot)
        seqs.dot <- apply(bpm.dot,1,paste,collapse="")
        names(seqs.dot) <- names(hseqs)
        return(seqs.dot)
    }
```

```
    DSFT <-
    function(nr,size,p.cut=0.002,conf=0.95){
        dsnr <- nr
        if(sum(nr)>size)
            dsnr <- round(nr/sum(nr)*size)
```

```
    thr <- qbinom(conf,size,p.cut)
    return(dsnr >= thr)
}
```

```
FAD <-
function(dst){
    if(class(dst)!="dist" & class(dst)!="matrix")
    stop("The input object must be of dist or matrix class.\n")
    return(sum(as.matrix(dst)))
}
```

```
fn.ab.1 <-
function(n,h=10000,r=0.5){
    if(class(n)!="numeric" & class(h)!="numeric" & class(r)!="numeric"){
        stop("All arguments must be numeric")}
    a <- floor(h*r^((1:n)-1))
    a[a<1] <- 1
    return(a)
}
```

```
fn.ab.2 <-
function(n,h=10000,r=3){
    if(class(n)!="numeric" & class(h)!="numeric" & class(r)!="numeric"){
        stop("All arguments must be numeric")}
    a <- floor(h*1/(1:n)^r)
    a[a<1] <- 1
    return(a)
}
```

```
fn.ab.3 <-
function(n,h=10000){
    if(class(n)!="numeric" & class(h)!="numeric"){
        stop("All arguments must be numeric")}
    a <- floor(h^(1/1:n))
    a[a<1] <- 1
    return(a)
}
```

```
FreqMat <-
function(seqs,nr=NULL){
    if(class(seqs)!="DNAStringSet" & class(seqs)!="AAStringSet")
        stop("The input object must be a DNAStringSet or AAStringSet \n")
    nt.nms <- DNA_BASES
```

```
    if (class(seqs)=="AAStringSet") nt.nms <- AA_STANDARD
    if (is.null(nr)) nr <- rep(1,length(seqs))
    if(length(seqs)!=length(nr))
        stop("The input objects must have the same length \n")
    strm <- as.matrix(seqs)
    res <-  apply(strm,2,function(x)
        tapply(nr,factor(x,levels=nt.nms),sum))
    colnames(res) <- 1:ncol(res)
    res[is.na(res)] <- 0
    return(res)
}
```

```
GenerateVars <-
function(seq,nhpl,max.muts,p.muts){
    if(class(seq)!="character")
    seq <- as.character(seq)
    if( !all(strsplit(seq,"")[[1]] %in% DNA_BASES))
        stop("The seq argument must be a DNA sequence")
    if(class(nhpl)!="numeric")
        stop("The nhpl argument must be numeric")
    if(length(p.muts)!=max.muts)
        stop("The p.muts argument must have the same length as max.muts")
    mutate <- function(nt){
        nt.nms <- DNA_BASES
        pnt <- rep(1/3,4)
        names(pnt) <- nt.nms
        fnt <- pnt; fnt[nt] <- 0
        return(sample(nt.nms,size=1,prob =fnt))
    }
    p.muts <- p.muts/sum(p.muts)
    ntv <- strsplit(seq,split="")[[1]]
    n.muts <- sample(max.muts,size=nhpl,prob=p.muts,replace=TRUE)
    len <-  length(ntv)
    vseqs <- character(nhpl)
    for(i in 1:nhpl){
        ipos <- sample(len,n.muts[i],replace=FALSE)
        nt.var <- sapply(ntv[ipos],mutate)
        mseq <- ntv
        mseq[ipos] <- nt.var
        vseqs[i] <- paste(mseq,collapse="")
    }
    return(vseqs)
}
```

```
geom.series <-
function(n,p=0.001){
    if(class(n)!="numeric" & class(p)!="numeric"){
```

```
            stop("All arguments must be numeric")}
        k <- 1:n
        return((1-p)^(k-1)*p)
    }
```

```
    GetInfProfile <-
    function(seqs,nr=NULL){
        if(is.null(nr)) nr <-rep(1,length(seqs))
        if(length(seqs)!=length(nr))
            stop("The input objects must have the same length \n")
        ct <- 2
        if(class(seqs)=="AAStringSet") ct <- log2(20)
        InfContent <- function(v){
            v <- v/sum(v)
            lgv <- ifelse(v==0,0,log2(v))
            return(ct+sum(v*lgv))
        }
        fm <- FreqMat(seqs,nr)
        return(apply(fm,2,InfContent))
    }
```

```
    GetQSData <-
    function(flnm,min.pct=0.1,type="DNA"){
        if(class(min.pct)!="numeric") stop("The min.pct argument must be numeric")
        lst <- ReadAmplSeqs(flnm,type)
        fl <- lst$nr/sum(lst$nr)*100 >= min.pct
        lst <- SortByMutations(lst$hseqs[fl],lst$nr[fl])
        return(list(seqs=lst$bseqs,nr=lst$nr,nm=lst$nm))
    }
```

```
    GetRandomSeq <-
    function(seq.len){
        if(class(seq.len)!="numeric") stop("The input must be numeric")
        nt.nms <- DNA_BASES
        return(paste(sample(nt.nms,seq.len,replace=TRUE),collapse=""))
    }
```

```
    GiniSimpsonMVUE <-
    function(w){
        if(class(w)!="numeric" & length(w)<=0)
            stop("The input object must be a numeric vector \n")
        p <- w/sum(w)
        pm1 <- (w-1)/(sum(w)-1)
        return(1 - sum(p*pm1))
```

```
}
```

```
GiniSimpson <-
function(w){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    n <- sum(w)
    if(n<2) return(NULL)
    p <- w/n
    return((1 - sum(p^2))*n/(n-1))
}
```

```
GiniSimpsonVar <-
function(w){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    n <- length(w)
    p <- w/sum(w)
    return(4/n*(sum(p^3)-sum(p^2)^2))
}
```

```
HCqProfile <-
function(w,q=NULL){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    if(is.null(q))
        q <- c(seq(0,0.9,0.1),seq(1,1.8,0.2),seq(2,3.75,0.25),
        seq(4,10,1),Inf)
    dv <- sapply(q,function(e) HCq(w,e))
    return(data.frame(q=q,HC=dv))
}
```

```
HCq <-
function(w,q){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    if(any(q<0)) stop("HCq numbers must be positive values")
    if(length(q)>1) {
        warning("Just the first q value is considered")
        q <- q[1]
    }
    if(q==0) return(length(w)-1)
    if(q==1) return(Shannon(w))
    if(q==Inf) return(0)
```

```
    p <- w/sum(w)
    return((1-sum(p^q))/(q-1))
}
```

```
HCqVar <-
function(w,q){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    if(any(q<0)) stop("HCq numbers must be positive values")
    n <- sum(w)
    if(n<2) return(NULL)
    p <- w/n
    return(1/n*(q/(q-1))^2*(sum(p^(2*q-1))-sum(p^q)^2))
}
```

```
HillProfile <-
function(w,q=NULL){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    if(is.null(q))
        q <- c(seq(0,0.9,0.1),seq(1,1.8,0.2),seq(2,3.75,0.25),
        seq(4,10,1),Inf)
    dv <- sapply(q,function(e) Hill(w,e))
    return(data.frame(q=q,qD=dv))
}
```

```
Hill <-
function(w,q){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector\n")
    if(any(q<0)) stop("Hill numbers must be positive\n")
    if(length(q)>1) {
        warning("Just the first q value is considered\n")
        q <- q[1]
    }
    if(q==0) return(length(w))
    if(q==1) return( exp(Shannon(w)))
    p <- w/sum(w)
    if(q==Inf) return(1/max(p))
    if(q==-Inf) return(1/min(p))
    return(sum(p^q)^(1/(1-q)))
}
```

```
IntersectStrandHpls <-
    function (nrFW , hseqsFW ,nrRV , hseqsRV , thr =0.001){
        if(length(nrFW)!= length(hseqsFW))
            stop("The length of the sequences and the counts must be equal \n")
        if(length(nrRV)!= length(hseqsRV))
            stop("The length of the sequences and the counts must be equal \n")
        if(class(hseqsFW)!= "character" & class(hseqsRV)!= "character" &
            class(hseqsFW)!= "DNAStringSet" & class(hseqsRV)!= "DNAStringSet" &
            class(hseqsFW)!= "AAStringSet" & class(hseqsRV)!= "AAStringSet")
        {stop("The sequences must be character vector or DNAStringSet or
            AAStringSet\n")}
        if(class(nrFW)!="numeric" & class(nrRV)!="numeric" &
            class(thr)!="numeric")
        {stop("The sequences must be numeric vector \n")}
        AlgnStrandHpls <-
            function (nrFW , hseqsFW ,nrRV , hseqsRV ){
                names(nrFW) <- as.character(hseqsFW)
                names(nrRV) <- as.character(hseqsRV)
                nms <- union(as.character(hseqsFW),as.character(hseqsRV))
                nb <- length(nms)
                pFW <- rep(0,nb)
                FWseq<-as.character(unlist(DNAStringSetList(hseqsFW)))
                idx <- which(nms %in% FWseq)
                pFW[idx] <- nrFW[nms[idx]]
                pRV <- rep(0,nb)
                RVseq<-as.character(unlist(DNAStringSetList(hseqsRV)))
                idx <- which(nms %in% RVseq )
                pRV[idx] <- nrRV[nms[idx]]
                return(list(pFW=pFW,pRV=pRV,Hpl=nms))
            }
        flFW <- nrFW /sum(nrFW) >= thr
        flRV <- nrRV /sum (nrRV) >= thr
        lst <- AlgnStrandHpls ( nrFW[flFW], hseqsFW[flFW],
                                nrRV[flRV], hseqsRV[flRV])
        fl <- lst $pFW > 0 & lst $pRV > 0
        hseqs <- lst$Hpl[fl]
        nr <- sum(lst$pFW[fl]+lst$pRV[fl])
        o <- order(nr, decreasing = TRUE)
        return(list(hseqs=DNAStringSet(hseqs[o]),nr=nr[o],pFW=lst$pFW,
                    pRV=lst$pRV))
    }
```

```
MutationFreq <-
function(dst=NULL,nm=NULL,nr=NULL,len=1){
    if(!is.null(dst)){
        if(class(dst)!="dist" & class(dst)!="matrix")
            stop("The input object must be dist or matrix class \n")
        nru <- rep(1,nrow(as.matrix(dst)))
        mf <- sum(as.matrix(dst)[1,]*nru)/sum(nru)
```

```
    } else {
        if(length(nm)!=length(nr))
            stop("The inputs nr and nm must have the same length \n")
    mf <- sum(nm*nr/sum(nr))/len
    names(mf) <- NULL
    }
    return(mf)
}
```

```
MutationFreqVar <-
function(nm,nr=NULL,len=1){
    if(is.null(nr)) nr <- rep(1,length(nm))
    if(!length(nm)==length(nr))
        stop("The inputs nr and nm must have the same length \n")
    N <- sum(nr)
    if(N<2) return(0)
    p <- nr/sum(nr)
    v <- ((sum(p*nm^2)-sum(p*nm)^2)/len^2) / N
    names(v) <- NULL
    return(v)
}
```

```
MutsTbl <-
function(hseqs,nr=NULL){
    if(is.null(nr))
        nr <- rep(1,length(hseqs))
    seq.tbl <- FreqMat(hseqs,nr)
    j <- apply(seq.tbl,2,function(x) which.max(x)[1])
    seq.tbl[cbind(j,1:ncol(seq.tbl))] <- 0
    return(seq.tbl)
}
```

```
NormShannon <-
function(w) {
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    h <- length(w)
    if(h<2) return(0)
    S <- Shannon(w)
    return(S/log(h))
}
```

```
NormShannonVar <-
function(w) {
```

```
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    h <- length(w)
    if(h<2) return(0)
    N <- sum(w)
    if(N<2) return(NULL)
    w <- w/N
    lgw <- ifelse(w==0,0,log(w))
    S <- -sum(w*lgw)
    return((sum(w*lgw^2)-S^2+(h-1)/(2*N)) / (log(h)^2*N))
}
```

```
NucleotideDiversity <-
function(dst,w=NULL){
    if(is.null(w))
        w <- rep(1,nrow(as.matrix(dst)))
    return(Rao(dst,w))
}
```

```
PolyDist <-
function(seqs,w=NULL){
    if(class(seqs)!="DNAStringSet" & class(seqs)!="AAStringSet")
        stop("The input object must be DNAStringSet or AAStringSet \n")
    if(is.null(w)) w <- rep(1,length(seqs))
    if(length(seqs)!=length(w))
        stop("The input objects must have the same length \n")
    seq.tbl <- FreqMat(seqs,w)
    nt <- sum(seq.tbl[,1])
    seq.tbl <- MutsTbl(seqs,w)
    seq.tbl <- seq.tbl[,apply(seq.tbl,2,function(x) sum(x)>0),drop=FALSE]
    return(colSums(seq.tbl)/nt)
}
```

```
RaoPowProfile <-
function(dst,w=NULL,q=NULL){
    if(class(dst)!="dist" & class(dst)!="matrix")
    stop("The input object must be of dist or matrix class  \n")
    if (is.null(w)) w<- rep(1,ncol(dst))
    if(nrow(as.matrix(dst))!=length(w))
        stop ("w and dst must have the same dimension")
    if(is.null(q))
        q <- seq(0,2,0.1)
    m <- length(q)
    dv <- sapply(1:m,function(i) RaoPow(dst,q[i],w))
    return(data.frame(q=q,qQ=dv))
}
```

```
RaoPow <-
function(dst,q,w=NULL){
    if(class(dst)=="matrix"){ dst<-as.dist(dst)}
    if(class(dst)!="dist")
        stop("The input object must be of dist or matrix class  \n")
    D <- as.matrix(dst)
    if (is.null(w)) w<- rep(1,attr(dst,"Size"))
    if (attr(dst,"Size")!=length(w))
        stop ("w and dst must have the same dimension")
    if(class(q)!="numeric") stop("The q input object must be numeric\n")
    if(length(q)>1){
        warning("Just the first q value is considered.\n")
        q <- q[1]
    }
    n <- sum(w)
    if(n<2) return(NULL)
    p <- w/n
    O <- p %*% t(p)   #  |p><p|
    res <- sum(D*O^q)
    return(res)
}
```

```
Rao <-
function(dst, w=NULL){
    if(class(dst)!="dist" & class(dst)!="matrix")
    stop("The input object must be of dist or matrix class \n")
    if (is.null(w)) w<- rep(1,ncol(dst))
    D <- as.matrix(dst)
    if(nrow(D)!=length(w)) stop ("w and dst must have the same dimension")
    n <- sum(w)
    if(n<2) return(0)
    p <- w/n
    return((n/(n-1)) * (t(p) %*% D %*% p))
}
```

```
RaoVar <-
function(dst,w=NULL){
    if(class(dst)!="dist") stop("The input object must be dist class  \n")
    if (is.null(w)) w<- rep(1,ncol(dst))
    if(nrow(as.matrix(dst))!=length(w))
        stop ("w and dst must have the same length")
    n <- sum(w)
    if(n<2) return(0)
    p <- w/n
    D <- as.matrix(dst)
    S <- -(p%*%t(p))
```

```
    diag(S) <- p*(1-p)
    return(4*t(p)%*%D%*%S%*%D%*%p/n)
}
```

```
ReadAmplSeqs <-
function(flnm,type="DNA"){
    if(type!="AA" & type!= "DNA") stop("Check the type input")
    if (type=="AA") seqs <- readAAStringSet(flnm)
    if (type=="DNA") seqs <- readDNAStringSet(flnm)
    nr <- sapply(names(seqs),function(str) strsplit(str,split="\\|")[[1]][2])
    nr <- as.numeric(nr)
    nr[is.na(nr)] <- 1
    return(list(nr=nr,hseqs=seqs))
}
```

```
Recollapse <-
function(seqs,nr){
    cls <- class(seqs)
    sqtbl <- sort(tapply(nr,as.character(seqs),sum),decreasing=TRUE)
    seqs <- names(sqtbl)
    names(seqs) <- 1:length(seqs)
    nr <- as.integer(sqtbl)
    if(cls=="DNAStringSet") seqs <- DNAStringSet(seqs)
    if(cls=="AAStringSet") seqs <- AAStringSet(seqs)
    return(list(nr=nr,seqs=seqs))
}
```

```
RenyiProfile <-
function(w,q=NULL){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    if(is.null(q))
        q <- c(seq(0,0.9,0.1),seq(1,1.8,0.2),seq(2,3.75,0.25),
        seq(4,10,1),Inf)
    dv <- sapply(q,function(e) Renyi(w,e))
    return(data.frame(q=q,renyi=dv))
}
```

```
Renyi <-
function(w,q){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    if(any(q<0)) stop("Renyi numbers must be positive values")
    if(length(q)>1){
```

```
            warning("Just the frisr q value is considered\n")
            q <- q[1]
        }
        if(q==0) return(log(length(w)))
        if(q==1) return(Shannon(w))
        p <- w/sum(w)
        if(q==Inf) return( -log(max(p)) )
        return(log(sum(p^q))/(1-q))
    }
```

```
    ReportVariants <-
    function(hseqs,ref.seq,nr=NULL,start=1){
        if(class(hseqs)!="DNAStringSet" & class(hseqs)!="AAStringSet")
            stop("The input object hseqs must be DNAStringSet  or AAString\n")
        if(is.null(nr)) nr<- rep(1,length(hseqs))
        if(class(nr)!= "numeric") stop("The imput object nr must be numeric \n")
        if(class(ref.seq)!="character" & class(ref.seq)!="DNAString"
        & class(ref.seq)!="AAString")
            {stop("The input object ref.seq must be character \n")}
        rnt <- strsplit(as.character(ref.seq),split="")[[1]]
        mnt <- as.matrix(hseqs)
        jdx <- which(sapply(1:ncol(mnt),function (j) sum(mnt[,j]!=rnt[j])>0))
        k <- 0
        vars <- data.frame(WT=character(),Pos=numeric(),Var=character(),
        Cov=numeric(),stringsAsFactors=FALSE)
        for(j in jdx){
            idx <- which(mnt[,j]!=rnt[j])
            vnr <- tapply(nr[idx],mnt[idx,j],sum)
            for (i in 1: length(vnr)){
                k <- k+1
                vars[k,"WT"]  <- rnt[j]
                vars[k,"Pos"] <- j+start-1
                vars[k,"Var"] <- names(vnr)[i]
                vars[k,"Cov"] <- vnr[i]
            }
        }
        return(vars)
    }
```

```
    SegSites <-
    function(seqs){
        if(class(seqs)!="DNAStringSet" & class(seqs)!="AAStringSet")
            stop("The input object must be a DNAStringSet or AAStringSet\n")
        return(sum( apply(FreqMat(seqs),2,function(x) sum(x>0)) > 1 ))
    }
```

```
Shannon <-
function(w){
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    h <- length(w)
    if(h<2) return(0)
    p <- w/sum(w)
    lgp <- ifelse(w==0,0,log(p))
    S <- -sum(p*lgp)
    if(all(w>=1)) S <- S+(h-1)/(2*sum(w))
    if(S>log(h)) S <- log(h)
    return(S)
}
```

```
ShannonVar <-
function(w) {
    if(class(w)!="numeric" & length(w)<=0)
        stop("The input object must be a numeric vector \n")
    h <- length(w)
    if(h<2) return(0)
    N <- sum(w)
    if(N<2) return(NULL)
    w <- w/N
    lgw <- ifelse(w==0,0,log(w))
    S <- -sum(w*lgw)
    return(((sum(w*lgw^2)-S^2) + (h-1)/(2*N)) / N)
}
```

```
SortByMutations <-
function(bseqs,nr){
    if(class(bseqs)!="DNAStringSet" & class(bseqs)!="AAStringSet")
        stop("The input object must be DNAStringSet or AAStringSet\n")
    if(length(bseqs)!=length(nr))
        stop("The input objects must have the same length \n")
    master <- bseqs[which.max(nr)]
    psa <- pairwiseAlignment(pattern=bseqs,subject=master)
    nm <- nmismatch(psa)
    tnm <- table(nm)
    o <- order(nm)
    bseqs <- bseqs[o]
    nr <- nr[o]
    nm <- nm[o]
    isq <- unlist(sapply(1:length(tnm),function(i) 1:tnm[i]))
    for(i in as.integer(names(tnm))){
        idx <- which(nm==i)
        o <- order(nr[idx],decreasing=TRUE)
        bseqs[idx] <- bseqs[idx[o]]
```

```
            nr[idx] <- nr[idx[o]]
        }
        frq <- round(nr/sum(nr)*100,2)
        nms <- paste("Hpl",nm,sprintf("%04d",isq),sep="_")
        names(bseqs) <- nms
        return(list(bseqs=bseqs,nr=nr,nm=nm))
    }
```

```
    SummaryMuts <-
    function(seqs,w=NULL,off=0){
        if(class(seqs)!="DNAStringSet" & class(seqs)!="AAStringSet")
            stop("The input object must be DNAStringSet or AAStringSet\n")
        if(length(seqs)<2){
            warning("More than 1 sequence is needed")
            return(NULL)
        }
        if(is.null(w)) w <- rep(1,length(seqs))
        pos.tbl <- FreqMat(seqs,w)
        mut.tbl <- MutsTbl(seqs,w)
        flags <- apply(mut.tbl,2,sum)>0
        pos <- which(flags)
        res <- data.frame(pos=pos+off,t(pos.tbl[,flags]))
        return(res)
    }
```

```
    TotalMutations <-
    function(hseqs,w=NULL){
        if(class(hseqs)!="DNAStringSet" & class(hseqs)!="AAStringSet")
            stop("The input object must be DNAStringSet or AAStringSet \n")
        if(is.null(w))
            w <- rep(1,length(hseqs))
        if(length(hseqs)!=length(w))
            stop("The input objects must have the same length \n")
        mut.tbl <- MutsTbl(hseqs,w)
        return(sum(apply(mut.tbl,1,function(x) sum(x))))
    }
```

```
    UniqueMutations <-
    function(hseqs){
        if(class(hseqs)!="DNAStringSet" & class(hseqs)!="AAStringSet")
            stop("The input object must be DNAStringSet or AAStringSet \n")
        mut.tbl <- MutsTbl(hseqs)
        return(sum(apply(mut.tbl,1,function(x) sum(x>0))))
    }
```
